CMR ENGINEERING COLLEGE

(Approved by AICTE-New Delhi, Affiliated to JNTUH)

Kondlakoya(v), Medchal Road, Hyderabad -501401

Department Of Computer Science & Engineering

DATABASE MANAGEMENT SYSTEMS LAB MANUAL

Class	:	II Year II Sem
Regulation	:	R16
A.Y.	:	2017-2018

Roadway Travels

"Roadway Travels" is in business since 1997 with several buses connecting different places in india. Its main office is located in Hydearabd.

The company wants to computerize its operations in the following areas:

- Reservation and Ticketing
- Cancellations

Reservation & Cancellation:

Reservations are directly handled by booking office. Reservations can be made 30 days in advance and tickets issued to passenger. One Passenger/person can book many tickets (to his/her family).

Cancellations are also directly handed at the bokking office.

In the process of computerization of Roadway Travels you have to design and develop a Database which consists the data of Buses, Passengers, Tickets, and Reservation and cancellation details. You should also develop query's usinf SQL to retrieve the data from database.

The above process involves many steps like

1. Analyzing the problem and identifying the Entities and Relationships,

2. E-R Model

3. Relational Model

4. Normalization

5. Creating the database

6. Querying.

Students are suppossed to work on these steps week wise and finally create a complete "Database System" to Roadway Travels. Examples are given at every experiment for guidance to students.

Experiment 1: E-R Model for "Roadway Travels" "Roadway Travels"

AIM :To Study And identify the entity in road way travels database system

Analyze the problem carefully and come up with the entities in it. Identify what data has to be persisted in the database. This contains the entities, attributes etc.

Entity:

An entity may be defined as a thing which is recognized as being capable of an independent existence and which can be uniquely identified. An entity is an abstraction from the complexities of some domain. When we speak of an entity we normally speak of some aspect of the real world which can be distinguished from other aspects of the real world. Entities can be thought of as nouns.

Examples:

A computer, an employee, a song, and a mathematical theorem.

The entities identified in "Roadway Travels" are :Bus, Ticket, Passenger

Relationship:

A relationship captures how two or more entities are related to one another. Relationships can be thought of as verbs, linking two or more nouns.

Examples:

An "owns" relationship between a company and a computer, a supervises relationship between an employee and a department, a performs relationship between an artist and a song, a proved relationship between a mathematician and a theorem.

The relationships identified in "Roadway Travels" are:

1. Reservation
2. Cancellation

Attributes:

Entities and relationships can both have attributes.

Examples:

An employee entity might have a Social Security Number (SSN) attribute; the proved relationship may have a date attribute.

The attributes identified for entities in “Roadway Travels” are:

1. Bus:
    a. BUS NUMBER,
    b. SOURCE,
    c. DESTINATION.
    d. NO OF DAYS AVAILBLE

2. Passenger:
    a. PASSEBGER _ID
    b. NAME,
    c. AGE,
    d. GENDER
    e. ADDRESS
3. Ticket:
    a. TICKET_ID
    b. JOUTNEY_DATE
    c. PNR NO
    d. SOURCE
    e. DESTINATION
    f. BUS NO

The attributes identified for relationships in “Roadway Travels” are:

1. Reservation:
    a. PNRNO
    b. JOUTNEY_DATE
    c. NUMBER_OF_SEATS

2. Cancellation:
    a. PNR _NO
    b. JOUTNEY_DATE
    c. NUMBER_OF_SEATS
    d.

Primary keys:

Bus NO. (Bus entity)

Passenger_id (passenger entity

Ticket_id (ticket entity).

E-R Diagrams:

Entity-relationship diagrams don't show single entities or single instances of relations. Rather, they show entity sets and relationship sets.

Example:

A particular song is an entity. The collection of all songs in a database is an entity set. The eaten relationship between a child and her lunch is a single relationship. The set of all such child-lunch relationships in a database is a relationship set. In other words, a relationship set corresponds to a relation in mathematics, while a relationship corresponds to a member of the relation.

Entity sets are drawn as rectangles, relationship sets as diamonds. If an entity set participates in a relationship set, they are connected with a line. Attributes are drawn as ovals and are connected with a line to exactly one entity or relationship set.

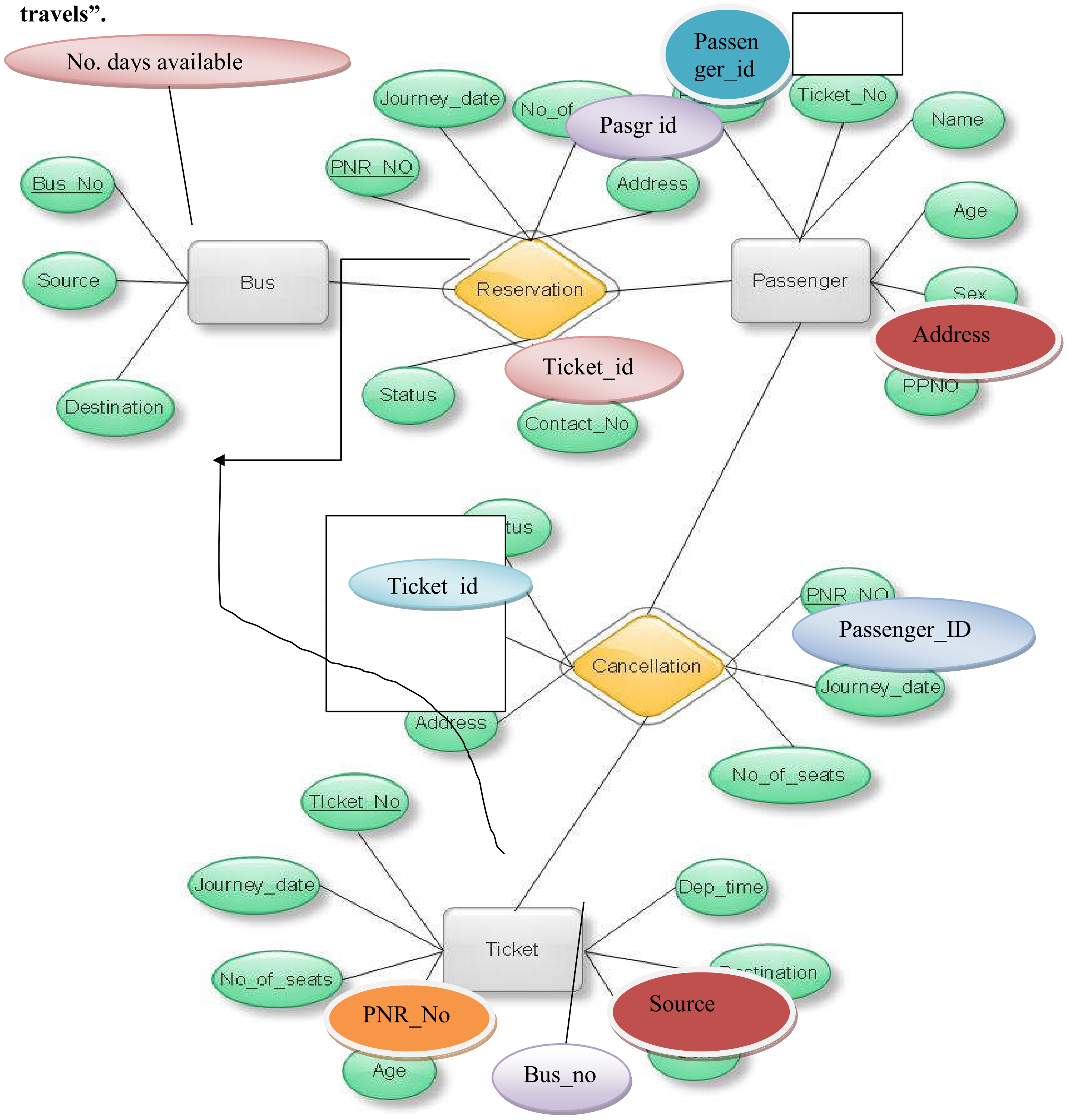
An underlined name of an attribute indicates that it is a key: two different entities or relationships with this attribute always have different values for this attribute.

Result: All the entities and relationships of database system are identified.

Experiment 2: Concept design with E-R Model for "Roadway Travels"

AIM : To draw the E-R diagram for Roadway Travels

The E-R Diagram for "Roadway travels".



Experiment 3
Relational Model "Roadway Travels" database

Aim : To perform the all the entities and relationships in tabular fashion

Description :There are different ways of representing relationships as tables based on the cardinality. Represent attributes as columns in table or as tables based on the requirement. Different types of attributes based on your E-R model.

The relational models for the entities and relations are as follows:

Bus:

Bus_No	Source	Destination	No of days available

Passenger:

P_ID	NAME	Age	Gender	Address	Ticket_ID

Ticket:

Ticket_ID	Journey_Date	PNR_NO	BUS_NO	Source	Destination

Reservation:

PNR_No	Journey_Date	No_of_seats

Cancellation:

PNR_No	Journey_Date	No_of_seats

Result:Relation model of database system is performed

Experiment 4
Normalization for database tables

AIM: To perform the normalization in a database system

Description: Database Normalisation is a technique of organizing the data in the database.

Normalization is a systematic approach of decomposing tables to eliminate data redundancy

Normalization rule are divided into following normal form.

1. First Normal Form
2. Second Normal Form
3. Third Normal Form
4. BCNF
5. Fourth Normal Form
6. Fifth Normal Form

To remove the redundancy from the following table

Passenger:

P_ID	NAME	Age	Gender	Address	Ticket_ID

After the normalization of above table

Passenger:

P_ID	NAME	Age	Gender	Address

Pastic

P_ID	Ticket_ID

Result: Normalization is performed in database system.

Experiment 5
Installation of MySQL
Creation & Alteration of tables using DDL commands

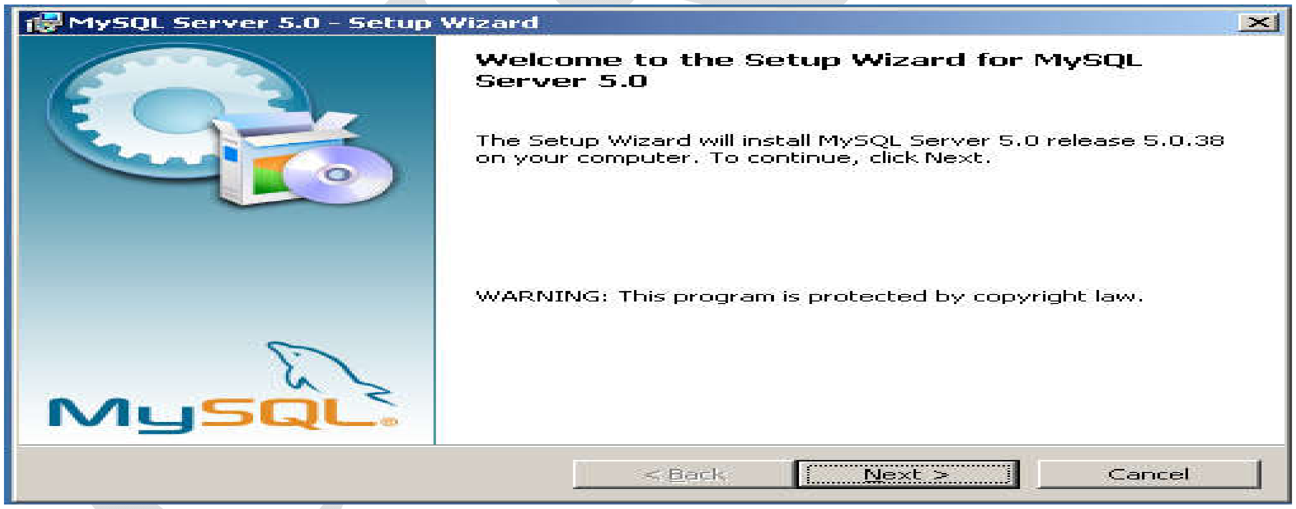
Aim:perform the installation of MYSQL and execute and manipulate all the DDL commands in a database system.

Installation of MySQL. In this week you will learn creating databases, creating tables, altering the databases, dropping tables and databases if not required. You will also try truncate, rename commands etc.

Installation of MYSQL.

Step 1: **Downloading and Starting the MySQL Installation Wizard**.

The process for starting the wizard depends on the contents of the installation package you download. If there is a setup.exe file present, double-click it to start the installation process.

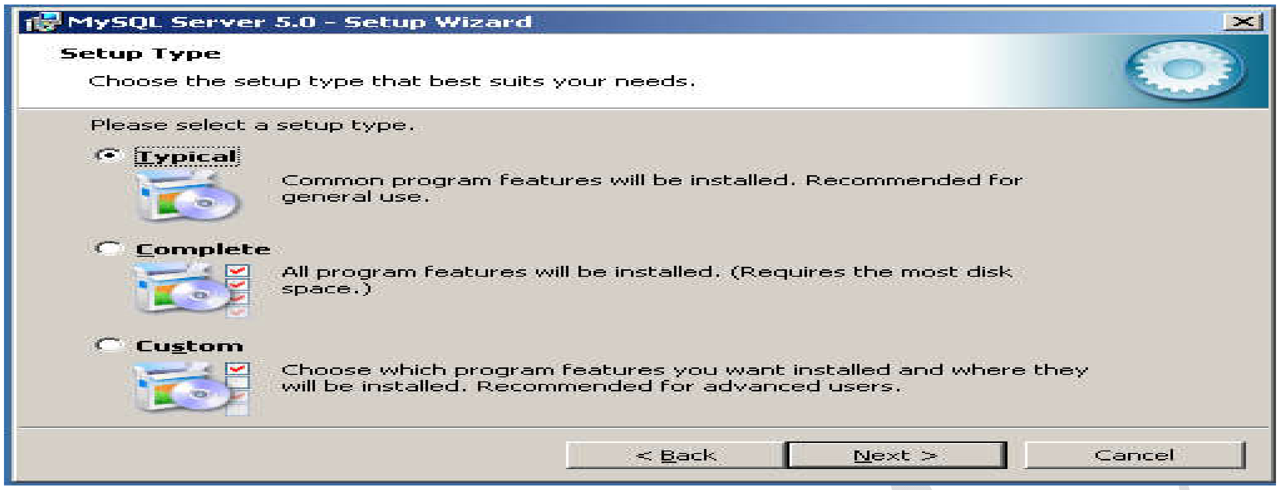


Welcome to the Setup Wizard for MySQL Server 5.0

The Setup Wizard will install MySQL Server 5.0 release 5.0.38 on your computer. To continue, click Next.

WARNING: This program is protected by copyright law.

Step 2: **Choosing an Installation Type.**
There are three installation types available: **Typical**, **Complete**, and **Custom**.



MySQL Server 5.0 - Setup Wizard

Setup Type

Choose the setup type that best suits your needs.

Please select a setup type.

● Typical
Common program features will be installed. Recommended for general use.

○ Complete
All program features will be installed. (Requires the most disk space.)

○ Custom
Choose which program features you want installed and where they will be installed. Recommended for advanced users.

The **Typical** installation type installs the MySQL server, the mysql command-line client, and the command-line utilities. The command-
line clients and utilities include mysqldump, myisamchk, and several other tools to help you manage the MySQL server.

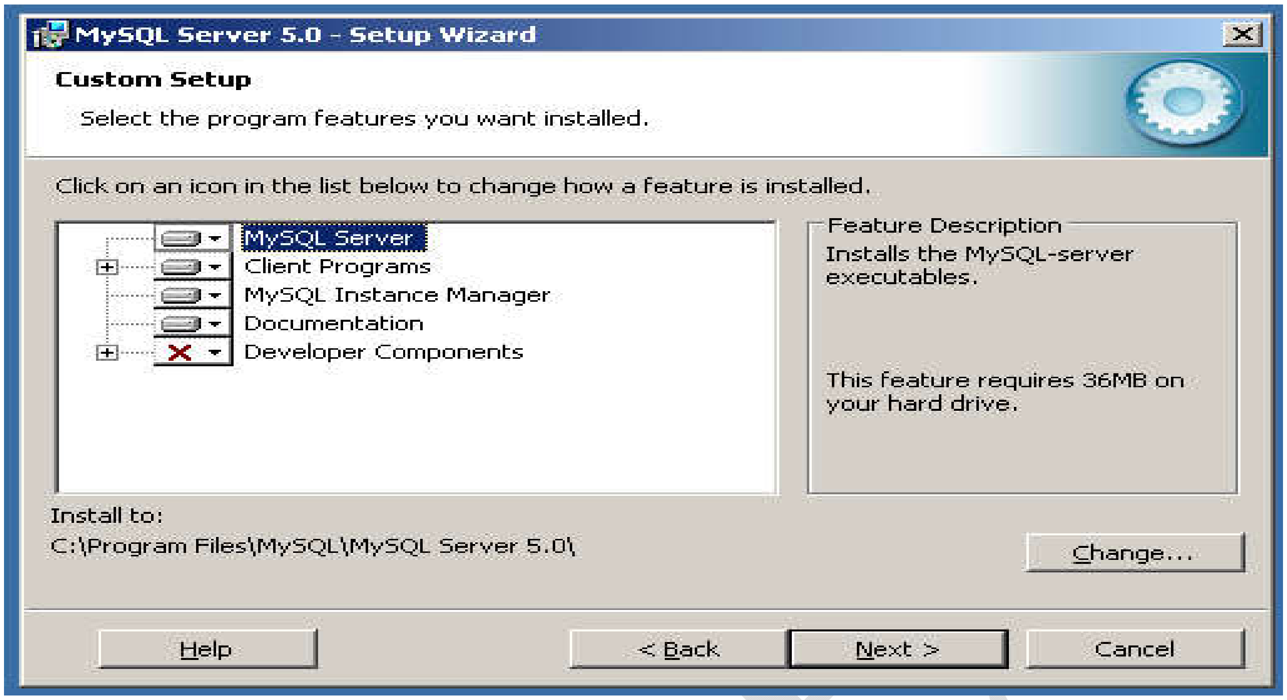
The **Complete** installation type installs all components included in the installation package. The full installation package includes components
such as the embedded server library, the benchmark suite, support scripts, and documentation.

The **Custom** installation type gives you complete control over which packages you wish to install and the installation path that is used.

If you choose the **Typical** or **Complete** installation types and click the NEXT button,

Step 3: **The Custom Installation Dialog.**

If you wish to change the installation path or the specific components that are installed by the MySQL Installation Wizard, choose the **Custom** installation type.


MySQL Server 5.0 - Setup Wizard

Custom Setup

Select the program features you want installed.

Click on an icon in the list below to change how a feature is installed.

- MySQL Server
- Client Programs
- MySQL Instance Manager
- Documentation
- Developer Components

Feature Description

Installs the MySQL-server executables.

This feature requires 36MB on your hard drive.

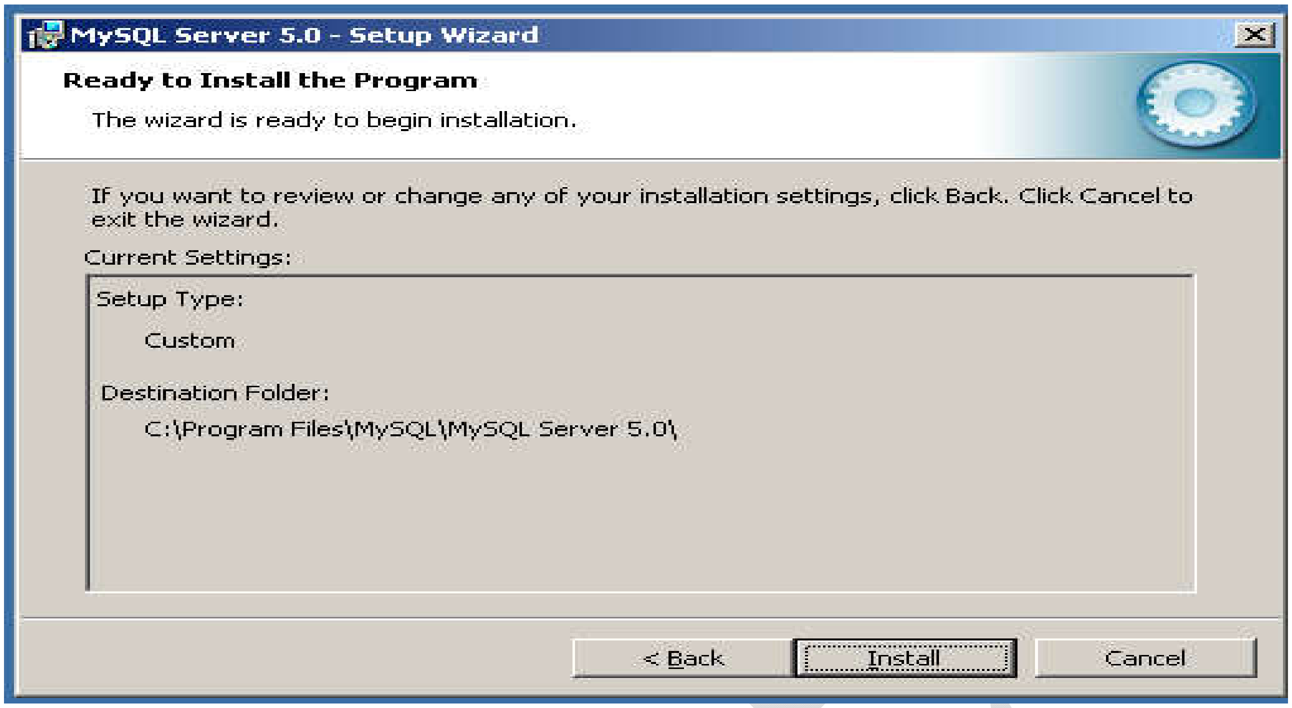
Install to:

C:\Program Files\MySQL\MySQL Server 5.0\

A tree view on the left side of the custom install dialog lists all available components. Components that are not installed have a red X icon; components that are installed have a gray icon. To change whether a component is installed, click that component's icon and choose a new option from the drop-down list that appears.

Step 4: The Confirmation Dialog.

Once you choose an installation type and optionally choose your installation components, you advance to the confirmation dialog. Your installation type and installation path are displayed for you to review.


MySQL Server 5.0 - Setup Wizard

Ready to Install the Program

The wizard is ready to begin installation.

If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.

Current Settings:

Setup Type:

Custom

Destination Folder:

C:\Program Files\MySQL\MySQL Server 5.0\

To install MySQL if you are satisfied with your settings, click the INSTALL button. To change your settings, click the BACK button. To exit the MySQL Installation Wizard without installing MySQL, click the CANCEL button.

And MySQL will be installed into your system automatically.

DESCRIPTION: A database system provides a data definition language (DDL) to specify the database schema. DDL also updates a special set of tables called the data dictionary or data directory**.**

Definitions:

1. **create:** ‘create’ command is used to define an SQL relation.

Syntax: create table table-name(field-name-1 data-type,
field-name-2 data-type,
…….
……
field-name-n data-type);

2. **desc:** ‘desc’ command is used to read the created relation.

Syntax: desc table-name;

3. **a) alter add: ‘**alter add’ command adds the attributes to an existing relation.

Syntax: alter table table-name add(field-name-1 data-type,……..);

b) alter modify: 'alter modify' command modifies the data types for the existing attributes.

Syntax: alter table table-name modify(field-name data-type);

4. **truncate:** 'truncate' command truncates the table i.e. all the rows will be deleted in the table.

Syntax: truncate table table-name;

5. **drop:** the 'drop' command deletes all information about the relation from the database i.e. it deletes the schema of the relation.

Syntax: drop table table-name;

<u>Creating tables:</u>

<u>Create table with following schma</u>

1. Bus:
   - i. BUS NUMBER,
   - ii. SOURCE,
   - iii. DESTINATION.
   - iv. NO OF DAYS AVAILBLE
2. Passenger:
   - i. PASSEBGER _ID
   - ii. NAME,
   - iii. AGE,
   - iv. GENDER
   - v. ADDRESS
3. Ticket:
   - i. TICKET_ID
   - ii. JOUTNEY_DATE
   - iii. PNR NO
   - iv. SOURCE
   - v. DESTINATION
   - vi. BUS NO
4. Pastic
   - i. PID,
   - ii. TID

The attributes identified for relationships in “Roadway Travels” are:

5. Reservation:
   i. PNRNO
   ii. JOUTNEY_DATE
   iii. NUMBER_OF_SEATS

6. Cancellation:
   a. PNR _NO
   b. JOUTNEY_DATE
   c. NUMBER_OF_SEATS

Primary keys:

BUS :Bus NO.

PASSENGER:Passenger_id

TICKET :Ticket_id

BUS Table:

```
SQL> CREATE TABLE  BUS
  (       BUS_NO VARCHAR2(10),
          SOURCE VARCHAR2(20),
          DESTINATION VARCHAR2(20),
          PRIMARY KEY (BUS_NO)
  );

SQL> Table created.

SQL>desc BUS;
```

Passenger table:

```
SQL> CREATE TABLE PASSENGER (PID VARCHAR(10) PRIMARY KEY,
         NAME VARCHAR(20) NOT NULL,
         AGE INTEGER,
         GENDER VARCHAR(5),
         ADDRESS VARCHAR(20));

TABLE CREATED.
```

```
SQL> desc PASSENGER;
```

Reservation Table:

```
SQL> CREATE TABLE RESERVATION (PNRNO INTEGER,
         JOURNEYDATE DATE,
         NOOFSEATS NUMBER)
 SQL> TABLE CREATED.

SQL>DESC RESERVATION;
```

Cancellation Table:

```
SQL>  CREATE TABLE  CANCELLATION
         PNRNO INTEGER,
         JOURNEYDATE DATE,
         NOOFSEATS NUMBER)

SQL> TABLE CREATED.

SQL>DESC CANCELLATION;
```

Ticket Table:

```
SQL> CREATE TABLE TICKET(TID NUMBER(15) PRIMARY KEY,
         JOURNDATE TIMESTAMP,
         SOURCE CHAR(10),
         DESTINATION CHAR(8),
         BUSNO VARCHAR2(10),
         PNRNO NUMBER(5),
         FOREIGN KEY(BUSNO) REFERENCES BUS1540(BUSNO));


SQL> TABLE CREATED.

SQL>desc TICKET;
```

To truncate a table:

Syntax is : truncate table table_name;

```
Ex: SQL> truncate table bus;
    SQL> Table Truncated.
```

To DROP a table:

Syntax is: drop table table_name;

```
Ex: SQL> Drop table bus;
    SQL> Table Dropped.
```

Result: installation of MYSQL and roadway travels tables are created is performed

Experiment 6

Inserting and modifying the data using DML commands

DML COMMANDS

AIM: To study and execute all the DML commands in a database system.

DESCRIPTION: A data manipulating language(DML) is a language that enables users to access or manipulate data as organized by the appropriate data model. They am basically two types:

1. Procedural DML.
2. Declarative DML.

The DML component of the SQL language is non-procedural. The DML commands in a database system are

DML commands are used for managing the data within schema objects. Some examples are.

I. SELECT- retrieve data from the database.
II. INSERT- insert data into a table.
III. Updates existing data within a table.

DELETE- delete all records from a table, the space for the records remain

Definitions:

1. insert : 'insert' command is used to load the data into the relation

   Syntax: insert into table-name(field-names) values (field-values);
2. delete: 'delete' command deletes only tuples .

   Syntax: delete from table-name;
3. update: 'update' command is used to update the existing data in the relation

   Syntax: update table-name set field-name—value where condition;

INSERT Command:

```
SQL> create table bus540(busno varchar2(10),source varchar2(20),
destination varchar2(20));

Table created.
```

```
SQL> insert into bus540 values('&busno','&source','&des');
Enter value for busno: 1234
Enter value for source: hyd
Enter value for des: guntur
old   1: insert into bus540 values('&busno','&source','&des')
new   1: insert into bus540 values('1234','hyd','guntur')

1 row created.

SQL> /
Enter value for busno: 5678
Enter value for source: guntur
Enter value for des: vizaq
old   1: insert into bus540 values('&busno','&source','&des')
new   1: insert into bus540 values('5678','guntur','vizaq')

1 row created.

SQL> /
Enter value for busno: 1345
Enter value for source: vinukonda
Enter value for des: guntur
old   1: insert into bus540 values('&busno','&source','&des')
new   1: insert into bus540 values('1345','vinukonda','guntur')

1 row created.


 SQL> update bus540 set des='kcp' where busno='1345';

1 row updated.


SQL> select * from bus540;

BUSNO              SOURCE     DES
-------------------- ---------- --------------------
1234            hyd       guntur
5678
    vizaq
1345            vinukonda  kcp

SQL> create table passenger1540(pid varchar(10) primary key,name varchar(20) not null,
age integer,gender varchar(5),address varchar(20));

Table created.
```

```
SQL> insert into passenger1540 values('&pid','&name',&age,'&gender','&address');
Enter value for pid: 111
Enter value for name: ruchin
Enter value for age: 10
Enter value for gender: m
Enter value for address: hyderabad
old   1: insert into passenger540 values('&pid','&name',&age,'&gender','&address')
new   1: insert into passenger540 values('111','ruchin',10,'m','hyderabad')

1 row created.

SQL> /
Enter value for pid: 222
Enter value for name: sunay
Enter value for age: 11
Enter value for gender: m
Enter value for address: kcp
old   1: insert into passenger540 values('&pid','&name',&age,'&gender','&address')
new   1: insert into passenger540 values('222','sunay',11,'m','kcp')

1 row created.

SQL> /
Enter value for pid: 333
Enter value for name: nari
Enter value for age: 27
Enter value for gender: m
Enter value for address: guntur
old   1: insert into passenger540 values('&pid','&name',&age,'&gender','&address')
new   1: insert into passenger540 values('333','nari',27,'m','guntur')

1 row created.

SQL> select * from passenger540;

PID        NAME                       AGE GENDE ADDRESS
---------- -------------------- ---------- ----- --------------------
111        ruchin                      10 m     hyderabad
222        sunay                       11 m     kcp
333        nari                        27 m     guntur


SQL> create table ticket1540(tid number(15) primary key,journdate timestamp,source
char(10),desti
nation char(8),busno varchar2(10),pnrno number(5),foreign key(busno) references
bus1540(busno));

Table created.
```

```
SQL> insert into ticket1540 values(&tid,'&journdate','&source','&destination','&busno',&pnrno);
Enter value for tid: 112233
Enter value for journdate: 15-mar-13 10:30:54
Enter value for source: hyd
Enter value for destination: chennai
Enter value for busno: 5678
Enter value for pnrno: 112
old   1: insert into ticket1540 values(&tid,'&journdate','&source','&destination','&busno',&pnrno
)
new   1: insert into ticket1540 values(112233,'15-mar-13 10:30:54','hyd','chennai','5678',112)

1 row created.


SQL> /
Enter value for tid: 223344
Enter value for journdate: 28-jul-13 9:30:33pm
Enter value for source: tirupathi
Enter value for destination: guntur
Enter value for busno: 1345
Enter value for pnrno: 114
old   1: insert into ticket1540 values(&tid,'&journdate','&source','&destination','&busno',&pnrno
)
new   1: insert into ticket1540 values(223344,'28-jul-13 9:30:33pm','tirupathi','guntur','1345',1
14)

1 row created.

SQL> /
Enter value for tid: 334455
Enter value for journdate: 29-mar-13 9:20:22pm
Enter value for source: manglore
Enter value for destination: hyd
Enter value for busno: 1234
Enter value for pnrno: 115
old   1: insert into ticket1540 values(&tid,'&journdate','&source','&destination','&busno',&pnrno
)
new   1: insert into ticket1540 values(334455,'29-mar-13 9:20:22pm','manglore','hyd','1234',115)

1 row created.

SQL> select * from ticket1540;

       TID          JOURNDATE                    SOURCE     DESTINAT   BUSNO
PNRNO

    112233     15-MAR-13 10.30.54.000000 AM     hyd         chennai      5678          112
```

```
   223344      28-JUL-13 09.30.33.000000 PM         tirupathi       guntur          1345
114

   334455      29-MAR-13 09.20.22.000000 PM         manglore       hyd             1234
115


SQL>  create table pastic11(tid integer,pid varchar(10),primary key(tid,pid),
foreign key(tid) references ticket540,foreign key(pid) references
passenger540);

Table created.

SQL> insert into pastic11 values(112233,'111');

1 row created.

SQL> insert into pastic11 values(1223344,'222');

1 row created.

SQL> insert into pastic11 values(223344,'333');

1 row created.
SQL> select * from pastic;

      TID         PID
---------- ----------
    112233       111
   1223344       222
    223344       333

SQL> create table reservation540(pnrno integer,journeydate date,noofseats number);

Table created.

SQL> insert into reservation540 values(&pnrno,'&date',&noofseats);
Enter value for pnrno: 123
Enter value for date: 15-mar-2013
Enter value for noofseats: 3
old   1: insert into reservation540 values(&pnrno,'&date',&noofseats)
new   1: insert into reservation540 values(123,'15-mar-2013',3)

1 row created.

SQL> /
Enter value for pnrno: 124
Enter value for date: 28-jul-2013
Enter value for noofseats: 2
old   1: insert into reservation540 values(&pnrno,'&date',&noofseats)
new   1: insert into reservation540 values(124,'28-jul-2013',2)

1 row created.

SQL> /
Enter value for pnrno: 111
Enter value for date: 29-mar-2013
Enter value for noofseats: 3
old   1: insert into reservation540 values(&pnrno,'&date',&noofseats)
new   1: insert into reservation540 values(111,'29-mar-2013',3)

1 row created.

SQL> select * from reservation540;

    PNRNO     JOURNEYDA   NOOFSEATS
        ---------- --------- ----------
      123       15-MAR-13         3
      124       28-JUL-13         2
      111       29-MAR-13         3

SQL> create table cance540(pnrno integer,journeydate date,noofseats number)

Table created.


SQL>insert into cance540(&pnrno,'&date','&noofseats');
Enter value for pnrno: 123
Enter value for date: 09-apr-2013
Enter value for noofseats: 2
old   1: insert into cance540 values(&pnrno,'&date',&noofseats)
new   1: insert into cance540 values(123,'09-apr-2013',2)

1 row created.

SQL> /
Enter value for pnrno: 111
Enter value for date: 21-dec-3013
Enter value for noofseats: 4
old   1: insert into cance540 values(&pnrno,'&date',&noofseats)
new   1: insert into cance540 values(111,'21-dec-3013',4)

1 row created.

SQL> select * from cance540;
```

```
    PNRNO     JOURNEYDA    NOOFSEATS
          ---------- --------- ----------
      123        09-APR-13          2
      111        21-DEC-13          4
```

SELECT Command:

Selecting values from the bus table.

```
SQL> SELECT * FROM BUS; (Selects all the attributes and display)
```

UPDATE Command:

```
SQL> UPDATE BUS SET bus_no=’boo4’ WHERE bus_no=’b003’;
```

Delete Command:

```
SQL> DELETE FROM BUS; (To delete entire rows from a table)
SQL> DELETE * FROM BUS WHERE bus_no=’boo1’; (To delete an individual row)
```

Result: Inserting of values into roadway travels tables are performed using with DML commands

Experiment 7

Queries using EXISTS, NOT EXIST, UNION, INTERSECT SET OPERATIONS

AIM: To execute all the set operations in a SQL language.

Description: If the answer to a query is a multiset of rows then we use the set operations. Set operations include:

1. Union
2. Intersect
3. Except

Definitions:

1. Union: RV S returns a relation instance containing all tuples that occur in either relation instance R or relation instance S (or both).
2. Intersect: R Λ S returns a relation instance containing all tuples that occur in both R and S.
3. Except: R-S returns a relation instance containing all tuples that occur in R but not in S.

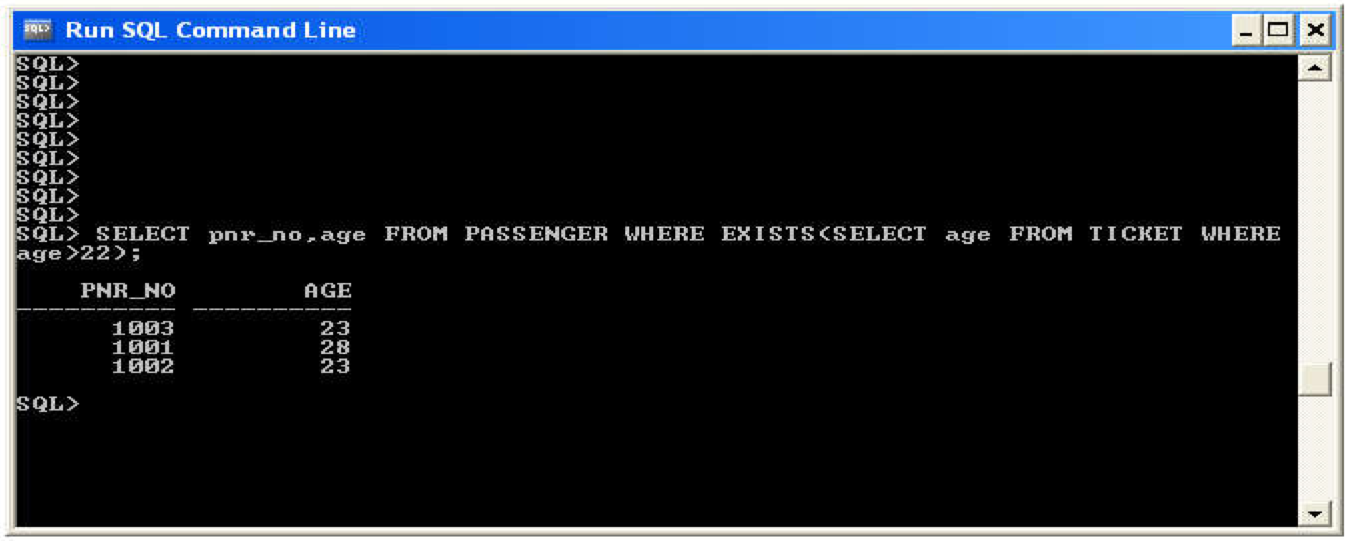
EXISTS Operator:

The EXISTS sub query is used when we want to display all rows where we have a matching column in both tables.

```
SQL> SELECT pnr_no,age FROM PASSENGER WHERE EXISTS(SELECT age FROM
TICKET WHERE age>22);
```

```
SQL> select name from passenger where exists (select * from ticket540 where tid='112233');

NAME
--------------------------------------------------
tarun
manu
deepak
shona
```

```
SQL> select name from passenger540 where exists (select * from ticket540 where tid='112233');

NAME
--------------------
ruchin
sunay
nari

SQL> select name from passenger540 where exists (select name from ticket540
 where tid='112233');

NAME
--------------------
ruchin
sunay
nari
```



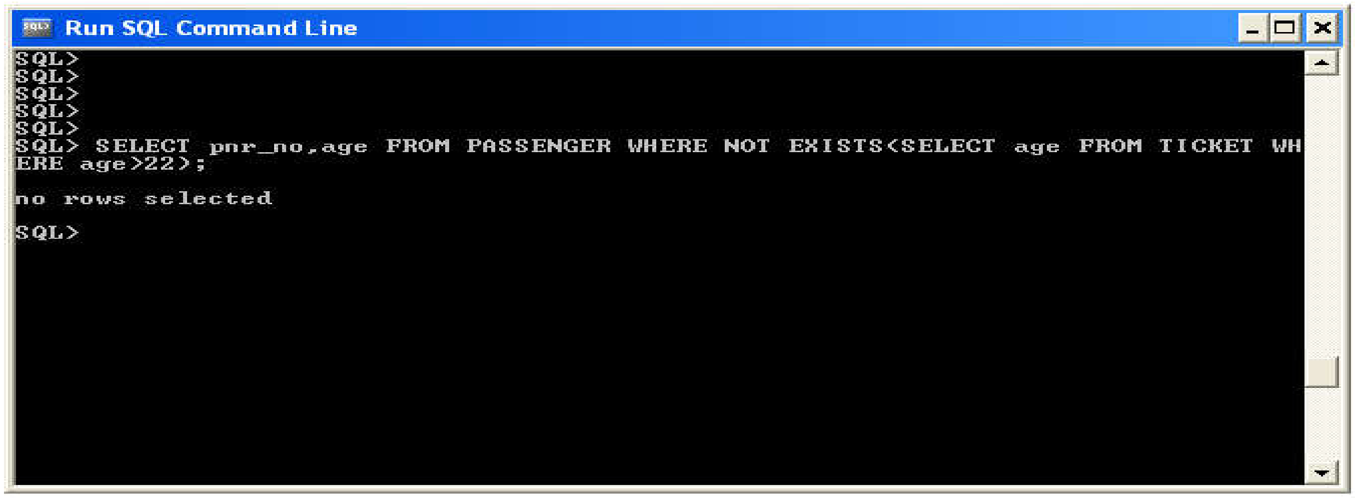
Run SQL Command Line

```
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL> SELECT pnr_no,age FROM PASSENGER WHERE EXISTS(SELECT age FROM TICKET WHERE
age>22);

    PNR_NO        AGE
---------- ----------
      1003         23
      1001         28
      1002         23

SQL>
```

NOT EXISTS Operator:

The NOT EXISTS subquery is used to display cases where a selected column does not appear in another table.

```
SQL> SELECT pnr_no,age FROM PASSENGER WHERE NOT EXISTS(SELECT age FROM
TICKET WHERE age>22);
```



```
SQL>
SQL>
SQL>
SQL>
SQL>
SQL> SELECT pnr_no,age FROM PASSENGER WHERE NOT EXISTS(SELECT age FROM TICKET WH
ERE age>22);

no rows selected

SQL>
```

SQL> (select name from passenger540 where address='hyderabad') minus (select name from passenger
where address='guntur');

```
NAME
--------------------
ruchin
```

SQL> (select name from passenger540 where address='kcp')minus (select name from passenger540 whe
ddress='guntur');

```
NAME
--------------------
```

UNION Operator:

Combine the results of two SELECT statements into one result set, and then eliminates and duplicate rows from that result set.

SQL> SELECT pnr_no FROM PASSENGER WHERE age>20 UNION SELECT pnr_no FROM RESERVATION WHERE no_of_seats > 2;

SQL> (select *from passenger540 where address='hyderabad')union (select *from passenger540
 where dress='kcp');

```
PID        NAME                  AGE GENDE ADDRESS
---------- -------------------- ---------- ----- --------------------
111        ruchin                10 m     hyderabad
```

```
222        sunay                    11 m     kcp

SQL> (select * from passenger540) union (select *from passenger540);

PID        NAME                       AGE GENDE ADDRESS
---------- -------------------- ---------- ----- --------------------
111        ruchin                     10 m     hyderabad
222        sunay                      11 m     kcp
333        nari                       27 m     guntur

SQL> (select * from passenger540) union all (select *from passenger540);

PID        NAME                       AGE GENDE ADDRESS
---------- -------------------- ---------- ----- --------------------
111        ruchin                     10 m     hyderabad
222        sunay                      11 m     kcp
333        nari                       27 m     guntur
111        ruchin                     10 m     hyderabad
222        sunay                      11 m     kcp
333        nari                       27 m     guntur

6 rows selected.

SQL> (select pid,name from passenger540)   union   (select pid,name from passenger540);

PID        NAME
---------- --------------------
111        ruchin
222        sunay
333        nari

SQL> (select pid,name from passenger540)union (select name,pid from passenger540);

PID                  NAME
-------------------- --------------------
111                  ruchin
222                  sunay
333                  nari
nari                 333
ruchin               111
sunay                222

6 rows selected.
```

INTERSECT Operator:

Combine the results of two SELECT statements into one result set, and returns only those rows that are returned by each of two statements.

```
SQL> >SELECT pnr_no FROM PASSENGER WHERE age>20 INTERSECT SELECT pnr_no
FROM RESERVATION WHERE no_of_seats > 2;


SQL> (select *from passenger540)intersect (select * from passenger540);

PID        NAME                    AGE GENDE ADDRESS
---------- -------------------- ---------- ----- --------------------
111        ruchin                  10 m     hyderabad
222        sunay                   11 m     kcp
333        nari                    27 m     guntur
```

Result: All the set operation queries are performed

Experiment 8

Queries using set comparison operators using ANY, ALL, IN and along with nested queries

AIM: To execute all the set comparison operations in a SQL language.

Description: If the answer to a query is a multiset of rows then we use the set operations. Set comparison operations include:

i. Any
ii. ALL
iii. In

Definitions:

The ANY Operator:the any operator allows you to specify multiple values in a where clause and select any value from that list

The All Operator:the all operator allows you to specify multiple values in a where clause and select all value from that list

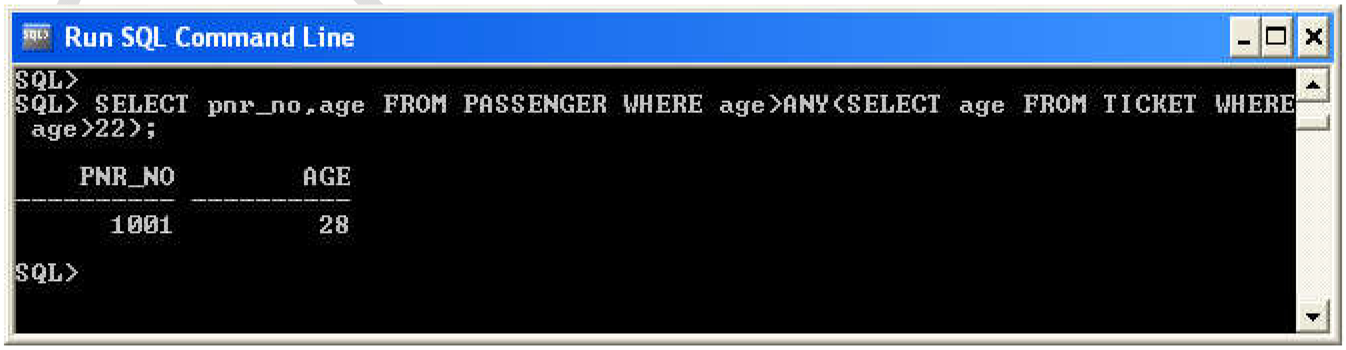
The IN Operator:the in operator allows you to specify multiple values in a where clause.

The BETWEEN Operator: The BETWEEN operator selects values within a range. The values can be numbers, text, or dates.

ANY Operator:

Compares values to each value returned by sub query.

SQL> SELECT pnr_no,age FROM PASSENGER WHERE age>ANY(SELECT age FROM TICKET WHERE age>22);

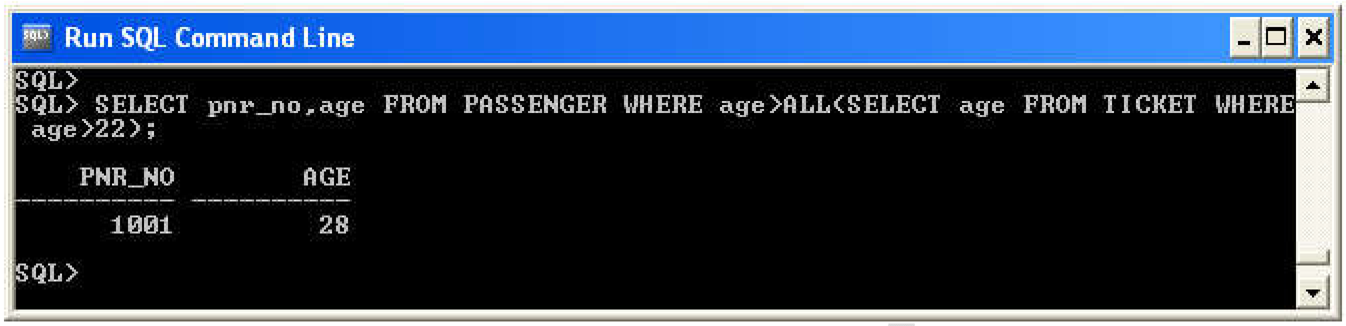


```
SQL>
SQL> SELECT pnr_no,age FROM PASSENGER WHERE age>ANY(SELECT age FROM TICKET WHERE
 age>22);

    PNR_NO        AGE
---------- ----------
      1001         28

SQL>
```

ALL Operator:

Compare values to every value returned by sub query.

SQL> SELECT pnr_no,age FROM PASSENGER WHERE age>ALL(SELECT age FROM TICKET WHERE age>22);

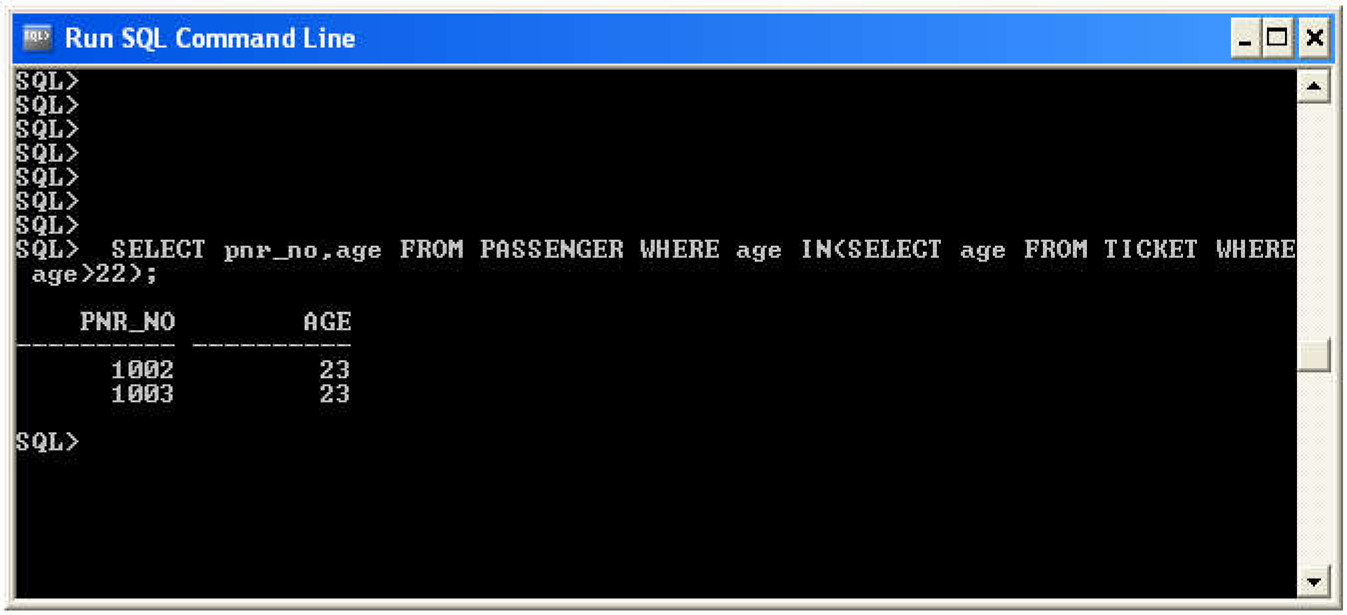


```
SQL>
SQL> SELECT pnr_no,age FROM PASSENGER WHERE age>ALL(SELECT age FROM TICKET WHERE
 age>22);

    PNR_NO        AGE
---------- ----------
      1001         28

SQL>
```

IN Operator:

Equal to any member in the list.

SQL> SELECT pnr_no,age FROM PASSENGER WHERE age IN(SELECT age FROM TICKET WHERE age>22);



```
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>  SELECT pnr_no,age FROM PASSENGER WHERE age IN(SELECT age FROM TICKET WHERE
 age>22);

    PNR_NO        AGE
---------- ----------
      1002         23
      1003         23

SQL>
```

BETWEEN Operator

SQL> SELECT name FROM PASSENGER WHERE age BETWEEN 30 AND 45;

1. Display unique pnr_no of all passengers.

SQL> SELECT DISTINCT pnr_no FROM PASSENGER;

2. Display all the names of male passengers.

SQL> SELECT name FROM PASSENGER WHERE sex = ’M’;

3. Display the ticket numbers and names of all passengers.

SQL> SELECT ticket_no, name FROM PASSENGER;

4. Display the source and destinations of all busses.

SQL> SELECT source, destination FROM BUS;

5. Find the ticket numbers of the passengers whose name starts with ‘A’ and ends with ‘H’.

SQL> SELECT ticket_no FROM PASSENGER WHERE name LIKE ‘A%H’;

6. Find the names of all the passengers whose age is between 30 and 45.

SQL> SELECT name FROM PASSENGER WHERE age BETWEEN 30 AND 45;

7. Display all the passengers names beginning with ‘A’.

SQL> SELECT name FROM PASSENGER WHERE name LIKE ‘A%’;

8. Display the sorted list of passenger’s names.

SQL> SELECT name FROM PASSENGER ORDER BY name ASC;

Result: All the set comparison operation queries are performed

Experiment 9

Queries using Aggregate functions (COUNT, SUM, AVG, MAX,and MIN) and Queries using GROUP BY, HAVING and Creation and droping of VIEWS.

AIM: To execute all the group functions and To execute all the group by Having clause and view in a SQL language.

Description: These can be applied on any column of a relation. Aggregate functions include:

1.Avg
2.Min
3.Max
4.Sum
5.Count

Definitions:

1. Sum: The input to sum must be a collection of numbers and it gives sum of those numbers.

Syntax: select Sum (field-name) from table-name;

2. Min: It presents the minimum value among tuples of a given attribute.

Syntax: select Min (field-name) from table-name;

3. Max: It presents the maximum value among tuples of a given attribute.

Syntax: select Max (field-name) from table-name;

4. Avg: It gives the average of the numbers.

Syntax: select Avg (field-name) from table-name;

5. Count: It counts the number of tuples in the relation.

Syntax: select Count (Distinct field-name) from table-name;

COUNT: It returns the number of rows in the query.

SQL> SELECT COUNT(*) FROM PASSENGER;

SUM: It returns the sum value of column.

SQL> SELECT SUM(ticket_no) FROM PASSENGER;

AVG: It returns the average value of column.

SQL> SELECT AVG(age) FROM PASSENGER;

MAX: It returns the maximum value of column.

SQL> SELECT MAX(age) FROM PASSENGER;

MIN: It returns the minimum value of column.

SQL> SELECT AVG(age) FROM PASSENGER;

Description: To write queries using clauses such as GROUP BY, HAVING and creation of VIEW, etc. and retrieving
information by joining tables.

Definitions:

GROUP BY CLAUSE: The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

SQL> SELECT name, AVG(age) FROM PASSENGER GROUP BY name;

HAVING CLAUSE: The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

SQL> SQL> SELECT name, AVG(age) FROM PASSENGER GROUP BY name HAVING AVG(age)>30;

Q1)WAQ to display infor pressent in reservation and cancellation tables using union operator

```
SQL>  select pnrno from reservation540 union select pnrno from cance540;

    PNRNO
----------
      111
      123
      124
```

Q)WAQ to find no of seats booked fro each pnrno using group by clause

```
SQL> select pnrno,sum(noofseats) from reservation540 group by pnrno;

    PNRNO NOOFSEATS
---------- --------------
      123            3
      124            2
      111            3
```

Q)WAQ to find no of pnrno's in a tableon which 123 pnr no is available using distinct

```
SQL> select count(distinct source)from bus540 where busno=1234;

COUNT(DISTINCTSOURCE)
---------------------
                 1
```

Q)WAQ to find distinct pnrno's that are present

```
SQL> select distinct pname from  reservation540;

    PNRNO
----------
      123
      124
      111
```

Q)WAQ to find total no of cancellation seats

```
SQL> select sum(noofseats) from cance540;

SUM(NOOFSEATS)
--------------
           6
```

A view is, in essence, a virtual table. It does not physically exist. Rather, it is created by a query joining one or more tables.

Creating VIEWS:

```
SQL> CREATE VIEW PASSENGERVIEW AS SELECT ticket_no,name,ppno,sex FROM
PASSENGER;

SQL> VIEW CREATED.

SQL> SELECT * FROM PASSENGERVIEW;
```

Updating a View

```
SQL> CREATE OR REPLACE PASSENGERVIEW AS SELECT ticket_no,ppno,sex from
PASSENGER;

SQL> VIEW CREATED.

SQL> SELECT * FROM PASSENGERVIEW;
```

Dropping a View

```
SQL> DROP VIEW PASSENGERVIEW;

SQL> VIEW DROPPED.
```

Result: All the Aggregate functions and GROUP BY ,HAVING AND VIEW queries are performed

Experiment 10

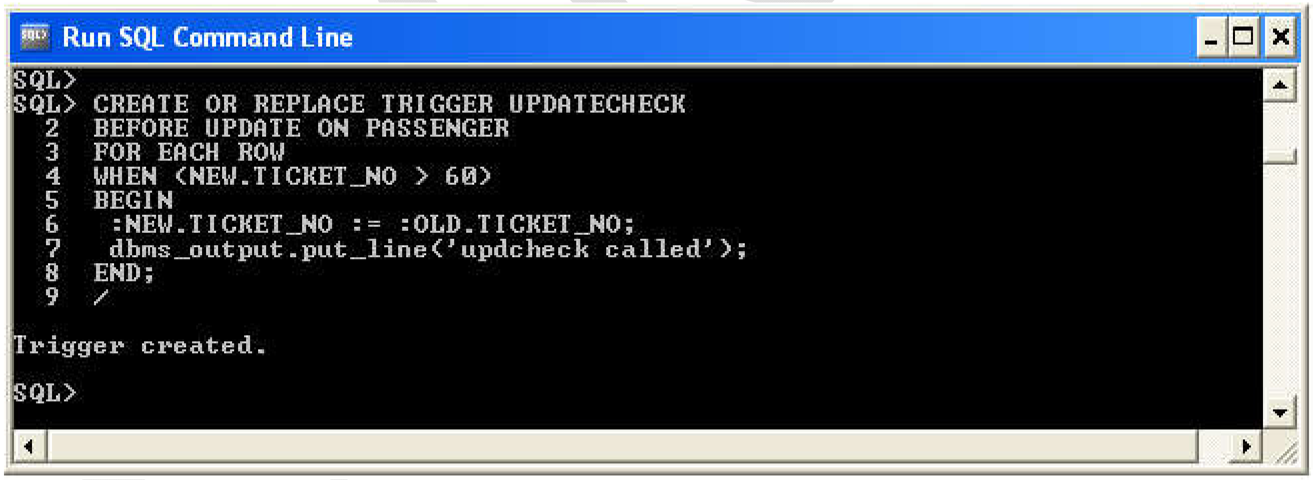
Creation of insert trigger, delete trigger, update trigger. Practice triggers using the roadway travels database.

Aim: To implement operations on relations using PL/SQL trigger

Definition: a **trigger** is a SQL procedure that initiates an action (i.e., fires an action) when an event (INSERT, DELETE or UPDATE) occurs. Since**triggers** are event-driven specialized procedures, they are stored in and managed by the **DBMS**.

```
SQL>CREATE OR REPLACE TRIGGER UPDATECHECK
BEFORE UPDATE ON PASSENGER
FOR EACH ROW
WHEN (NEW.TICKET_NO > 60)
BEGIN
 :NEW.TICKET_NO := :OLD.TICKET_NO;
 dbms_output.put_line('updcheck called');
END;
/
```

OUT PUT:



```
SQL>
SQL> CREATE OR REPLACE TRIGGER UPDATECHECK
  2  BEFORE UPDATE ON PASSENGER
  3  FOR EACH ROW
  4  WHEN (NEW.TICKET_NO > 60)
  5  BEGIN
  6   :NEW.TICKET_NO := :OLD.TICKET_NO;
  7   dbms_output.put_line('updcheck called');
  8  END;
  9  /

Trigger created.

SQL>
```

Result: A triggers in PL/SQL language are performed.

Experiment 11

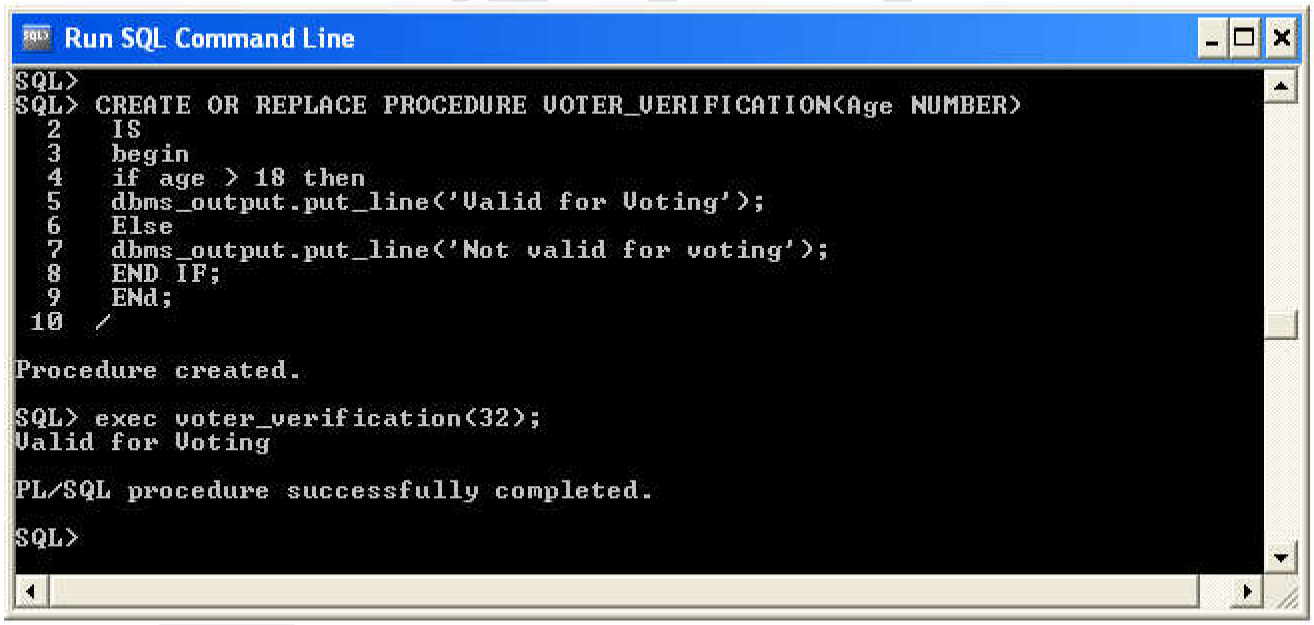
Creation of stored procedure, Execution of procedure and modification of procedure. Practice procedures using the database.

Aim: To implement Procedure on relations using PL/SQL

Definition: A **procedure** is a subprogram that performs a specific action.

```
CREATE OR REPLACE PROCEDURE VOTER_VERIFICATION(Age NUMBER)
 IS
 begin
 if age > 18 then
 dbms_output.put_line('Valid for Voting');
 Else
 dbms_output.put_line('Not valid for voting');
 END IF;
 ENd;
```

OUT PUT:


Run SQL Command Line

```
SQL>
SQL> CREATE OR REPLACE PROCEDURE VOTER_VERIFICATION(Age NUMBER)
      IS
      begin
      if age > 18 then
      dbms_output.put_line('Valid for Voting');
      Else
      dbms_output.put_line('Not valid for voting');
      END IF;
      ENd;
     /

Procedure created.

SQL> exec voter_verification(32);
Valid for Voting

PL/SQL procedure successfully completed.

SQL>
```

Result: A Procedure in PL/SQL language are performed.

Experiment 12

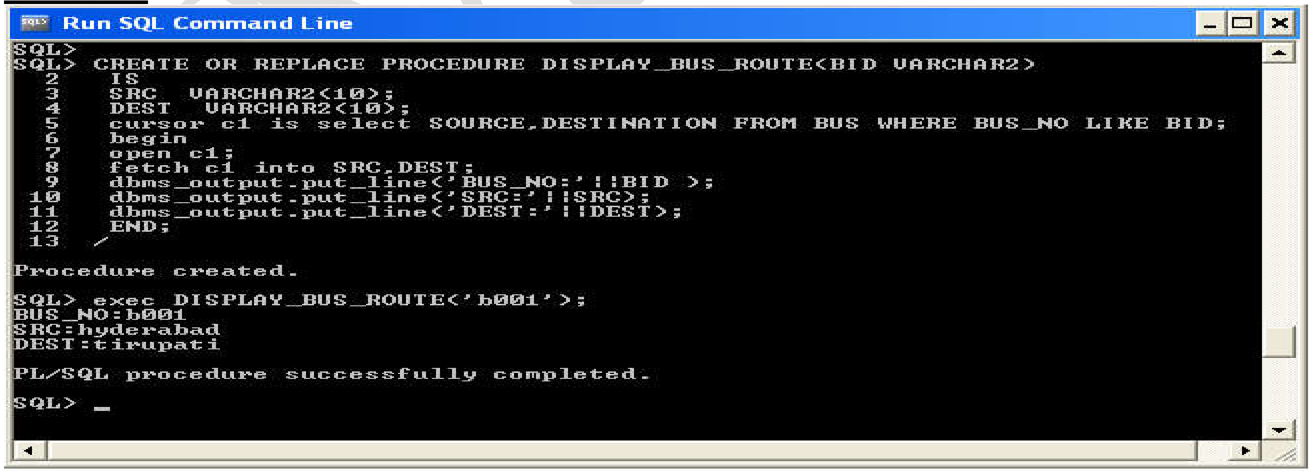
Declare a curser that defines a result set. Open the curser to establish a the result set. Fetch the data into local variables as needed from the cursor, one row at a time. Close the cursor when done.

Aim: To implement cursors on relations using PL/SQL

Definition: A cursor is a temporary work area created in the system memory when a SQL statement is executed. A cursor contains information on a select statement and the rows of data accessed by it.

```
SQL> CREATE OR REPLACE PROCEDURE DISPLAY_BUS_ROUTE(BID VARCHAR2)
 IS
 SRC  VARCHAR2(10);
 DEST  VARCHAR2(10);
 cursor c1 is select SOURCE,DESTINATION FROM BUS WHERE BUS_NO LIKE BID;
 begin
 open c1;
 fetch c1 into SRC,DEST;
 dbms_output.put_line('BUS_NO:'||BID );
 dbms_output.put_line('SRC:'||SRC);
 dbms_output.put_line('DEST:'||DEST);
 END;
```

OUTPUT:



```
SQL>
SQL> CREATE OR REPLACE PROCEDURE DISPLAY_BUS_ROUTE(BID VARCHAR2)
  2    IS
  3    SRC   VARCHAR2(10);
  4    DEST   VARCHAR2(10);
  5    cursor c1 is select SOURCE,DESTINATION FROM BUS WHERE BUS_NO LIKE BID;
  6    begin
  7    open c1;
  8    fetch c1 into SRC,DEST;
  9    dbms_output.put_line('BUS_NO:'||BID );
 10    dbms_output.put_line('SRC:'||SRC);
 11    dbms_output.put_line('DEST:'||DEST);
 12    END;
 13  /

Procedure created.

SQL> exec DISPLAY_BUS_ROUTE('b001');
BUS_NO:b001
SRC:hyderabad
DEST:tirupati

PL/SQL procedure successfully completed.

SQL> _
```

Result: A cursors in PL/SQL language are performed.